Rediseño de la arquitectura de un personalizador para ecommerce: una revisión del proceso de migración de tecnología e integración de patrones de diseño de software.

Stiven Guerra Chaverra

Informe de práctica presentado para optar al título de Ingeniero de Sistemas

Asesor
Diana Margot López Herrera, MG en Ingeniería

Universidad de Antioquia
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas
Medellín
2024

Agradecimientos

Quiero expresar mis agradecimientos a mis más cercanos amigos y compañeros de carrera, pues sin ellos no sería quien soy hoy día, ni habría logrado llegar hasta este punto: A Sebastián Gómez, por ayudar a forjar mi carácter al nunca dudar en expresarme sus opiniones sobre mi y mis acciones cuando lo pensaba apropiado, siempre de las formas más sinceras y acertadas con la más pura intención de ayudar a mejorar. A Ian Marcos Ortiz, pues fue la mejor compañía que pude tener a través de los múltiples proyectos grupales que afrontamos, siempre proporcionando la confianza y tranquilidad de que podía confiar en él para resolver cualquier reto académico. Y a Anderson Villa, quien me demostró que está bien confiarle los sentimientos a los amigos y resolver conflictos que se puedan presentar en las relaciones.

En último lugar, pero tal vez el más importante, le brindo mis más inmensos agradecimientos a mi maestra, jefa y asesora, Diana Margot López Herrera, quien me abrió puertas al mundo laboral y me guió de la mejor manera posible, pues no solo me dió lecciones de cómo redactar correos y estructurar propuestas de proyecto sino que también me compartió su experiencia en la vida para aprender a afrontar obstáculos personales y académico.

Por todo esto y mucho más, les agradezco por haberme ayudado, pues solo puedo decir que "fui ayudado" porque me ayudaron mucho más de lo que podría mencionar.
Gracias.

Tabla de contenido

LISTA DE FIGURAS

RESUMEN

Este proyecto aborda la necesidad urgente de mejorar la mantenibilidad de una aplicación web de comercio electrónico para la personalización de productos de un cliente, aplicación la cuál es proveída, gestionada y mantenida por CreateMe Technologies, empresa en que se desarrolla la práctica. Al ser esta pobremente actualizada desde el 2013, a día de hoy enfrenta dificultades de mantenimiento debido a la acumulación de código sin prácticas estandarizadas ni documentación adecuada. Pese a que este problema no afecta la funcionalidad, sí implica altos costos de mantenimiento, errores frecuentes y riesgos de respuesta tardía. La solución propuesta consiste en desarrollar una nueva versión de la aplicación realizando una migración a una tecnología más moderna, lo cual facilitará la aplicación de una arquitectura rediseñada, implementando patrones de diseño y prácticas de código limpio. Los objetivos incluyen diagnosticar fallas actuales, identificar oportunidades de mejora, diseñar y desarrollar una nueva aplicación con tecnologías modernas y crear documentación exhaustiva. Este trabajo se apoya de obras escritas y respetadas sobre principios de legibilidad, mantenibilidad y patrones de diseño para asegurar un código escalable y sostenible. El trabajo se llevó a cabo con una metodología que comprende dos fases: una inicial de análisis y conceptualización, y una de desarrollo utilizando la metodología RAD, enfocada en ciclos rápidos e iterativos. Al hacer uso de gestión de versiones y documentación clara, el proyecto busca entregar una aplicación robusta, flexible y de alta calidad, que aumente la productividad y reduzca errores, asegurando su mantenibilidad y adaptabilidad a largo plazo.

Palabras clave **— Migración de Tecnología, Rediseño de Software, Arquitectura, Patrones de Diseño, Calidad de Software.**

ABSTRACT

This project addresses the urgent need to improve the maintainability of a web application for product customization for a client, which is provided, managed, and maintained by CreateMe Technologies. Since the application has been poorly updated since 2013, it now faces maintenance challenges due to the accumulation of code with no standardized practices or adequate documentation. Although this problem does not affect functionality, it does result in high maintenance costs, frequent errors, and risks of delayed responses. The proposed solution is to develop a new version of the application by migrating it to a modern technology, which will facilitate the implementation of a redesigned architecture, incorporating design patterns and clean code practices. The objectives include diagnosing current issues, identifying improvement opportunities, designing and developing a new application with modern technologies, and creating comprehensive documentation. This work is supported by well-established literature on principles of readability, maintainability, and design patterns to ensure scalable and sustainable code. The work was carried out using a methodology that consists of two phases: an initial analysis and conceptualization phase, and a development phase using the RAD methodology, focused on rapid and iterative cycles. By making use of version control, and clear documentation, the project aims to deliver a robust, flexible, and high-quality application that increases productivity and reduces errors, ensuring long-term maintainability and adaptability.

Keywords **— Technology Migration, Software Redesign, Architecture, Design Patterns, Software Quality.**

I. INTRODUCCIÓN

En la actualidad existen incontables cantidades de aplicaciones, cada una con funcionalidades y propósitos diferentes, algunas más modernas que otras y algunas funcionales pero obsoletas debido al pobre mantenimiento que se les ha brindado. Las aplicaciones cuyo código no se cuida adecuadamente tienden a volverse inmantenibles con el tiempo. Este fenómeno, conocido como "deuda técnica", ocurre cuando se priorizan soluciones rápidas por encima de buenas prácticas de codificación, acumulando problemas que hacen que el mantenimiento y la evolución del software sean cada vez más difíciles y costosos. La falta de refactorización regular puede resultar en un código enmarañado, difícil de entender y propenso a errores, lo cual deriva en incrementar el tiempo necesario para implementar nuevas funcionalidades y corregir defectos. Considerar desde un inicio el desarrollo de aplicaciones mantenibles es crucial, pues facilita la adaptación a nuevos requisitos, mejora la productividad del equipo de desarrollo y reduce el riesgo de introducir errores al realizar cambios. La refactorización es una práctica clave para mantener la calidad del código y asegurar que las aplicaciones puedan evolucionar de manera sostenible y eficiente [1]. A partir de esta noción se propuso el presente proyecto, el cual buscó atender una urgente necesidad de mejorar la mantenibilidad de una aplicación web de e-commerce, que es el customizador de productos para uno de los clientes más importantes y relevantes para CreateMe. Problema el cual es en gran parte causado por la antigüedad de la aplicación, cuyo origen data del 2013, y que a lo largo de los años, los múltiples desarrolladores que han pasado por el proyecto y junto a la falta de estándares y documentación bien definidos, se ha reiterado en seguir construyendo sobre una fuente de código que es difícil de entender y que con cada trabajo se hace aún más difícil de mantener.

Pese a que el estado de la aplicación no ha comprometido el funcionamiento de la misma, el verdadero problema se presenta a nivel del negocio, pues tener una aplicación poco mantenible incurre constantemente en altos costos y tiempos en la implementación de nuevas funcionalidades, además de la frecuente aparición de bugs debido al poco control que se tiene sobre la calidad de la aplicación. Y en última instancia, pero no menos importante, el latente riesgo de que al faltar las personas con el suficiente conocimiento para atender requisitos o problemas urgentes, no se pueda tener un tiempo de acción tan inmediato como se requiere. De

esta forma se expone el principal problema que se busca solucionar y se proponen métodos para atender la situación a través del conocimiento técnico que se ha creado a lo largo de los años por figuras importantes en el mundo de la Ingeniería de Software.

Si bien es de saber que el desarrollo de aplicaciones web ha evolucionado considerablemente y que con ello, las expectativas de calidad, rendimiento y mantenibilidad se han vuelto más demandantes, el cumplir con altos estándares de calidad es esencial, y fundamentar el proyecto en principios y prácticas sólidas de desarrollo de software será la forma de dar solución al problema expuesto. Siendo la propuesta inicial para la solución, la creación de una nueva versión de la aplicación existente, cuyo factor diferencial será la migración a una tecnología moderna que facilite el rediseño de la arquitectura y la implementación de patrones de diseño y prácticas de código limpio, representando así un reinicio fresco y completo para la aplicación, en un entorno moderno y con una calidad de código mucho más alta.

II. OBJETIVOS

A. Objetivo general

Desarrollar una versión mejorada del personalizador para ecommerce a partir de un rediseño de la arquitectura y la implementación de buenas prácticas de calidad de software, para mejorar la mantenibilidad de esta.

B. Objetivos específicos

1. Diagnosticar las fallas de implementación de buenas prácticas que incurren en la obsolescencia del código fuente.
2. Definir las buenas prácticas y la arquitectura que deben ser incorporadas en la nueva versión del personalizador para ecommerce.
3. Diseñar una nueva aplicación que implemente la arquitectura y los patrones de diseño definidos.
4. Desarrollar una nueva aplicación implementando tecnologías modernas y los estándares definidos.
5. Desplegar y presentar al equipo de TI de la empresa para validar el resultado como una mejora significativa para la aplicación.

III. MARCO TEÓRICO

El primer problema a afrontar fue la ilegibilidad y poca mantenibilidad del código actual de la aplicación, el cual se originó de la constante construcción de esta sin una consistente y adecuada aplicación de estándares y prácticas desde un principio. Por lo cual, al desarrollar la nueva aplicación para que pueda ser mantenida y actualizada a lo largo del tiempo, se tienen en cuenta múltiples principios que van más allá de solo permitir que la aplicación funcione a los ojos de un usuario final, sino que a nivel de desarrollo, permita su escalabilidad sin sacrificar su mantenibilidad. La guía esencial que se tiene en cuenta para escribir código que no sólo funcione, sino que sea fácil de leer, mantener y mejorar, enfatiza en la importancia de la simplicidad, la legibilidad y la refactorización continua. Al aplicar estos principios, se buscaba que el código fuente de la nueva aplicación fuese comprensible para cualquier desarrollador que lo lea, facilitando así la colaboración y el mantenimiento, pues la claridad en la escritura del código reduce la probabilidad de errores por causa humana y mejora la eficiencia en el desarrollo de nuevas funcionalidades [2].

Por otro lado, al trabajar en una aplicación tan importante para la empresa, la cual se espera continuar mejorando y construyendo a lo largo de los años, se tuvieron en cuenta ciertos principios que mejoren la calidad del trabajo no sólo en términos del resultado, sino con respecto al proceso mismo. Para esto se consideró un enfoque práctico y ágil para el desarrollo de software, lo cual se refiere al uso de mejores prácticas esenciales para cualquier desarrollador que aspire a producir software de alta calidad en plazos de tiempo prudentes. Al hablar de estos principios que mejoran los procesos se encuentra la importancia de la automatización, la gestión de versiones, las pruebas continuas y la documentación clara. Implementar estas estrategias facilita la creación de una aplicación robusta y flexible que pueda ser activamente construída sin tener que incurrir en reprocesos o errores inadvertidos [3]. Por ejemplo, la automatización de pruebas y despliegues asegurará que cualquier cambio en el código sea probado de manera efectiva, reduciendo el riesgo de introducir errores. La gestión de versiones facilitará el control de cambios y la colaboración entre desarrolladores, mientras que una documentación clara permitirá que cualquier miembro del equipo comprenda el funcionamiento y las características de la aplicación.

Finalmente, es de vital importancia considerar que crear un buen producto de código que sea mantenible y escalable, no será el resultado de únicamente escribir un código leíble y de llevar a cabo procesos eficientes y bien registrados, también se debe invertir esfuerzo en un trabajo de diseño, a través del cual se busque hacer uso de patrones que permitan definir cómo funcionan los componentes de una aplicación y cómo estos interactúan entre sí. Esto gracias a que los patrones de diseño son soluciones probadas a problemas comunes en el desarrollo de software, que proporcionan un lenguaje común para los desarrolladores [5]. La aplicación de estos patrones en el presente proyecto permite crear una arquitectura bien estructurada y modular, haciendo posible que la aplicación como un todo no solo sea legible en su código o que los procesos realizados sean fáciles de seguir y comprender, sino que además esté estructurada de tal forma que nuevas implementaciones, cambios o en su defecto, la aparición de errores, no comprometen el negocio al requerir extensas cantidades de trabajo o que incluso lleguen a ser imposibles debido a la arquitectura misma.

Tras introducir los conceptos que se deben implementar, el siguiente paso lógico es definir cómo implementarlos. Uno de los procesos más importantes y comunes en el mantenimiento de aplicaciones de software es la *refactoring*, el cual consiste de realizar cambios en el código fuente de tal forma que se conserve la funcionalidad, pero se modifique su estructura en pos de que este sea más fácil de entender, manipular y/o mantener. Se considera hacer uso del *refactoring* para el proyecto en cuestión, pues esto permitiría hacer uso de gran parte de las funcionalidades existentes y simplemente llevar a cabo modificaciones en aquellas secciones en donde sea imperativo implementar mejoras, permitiendo así invertir la mayor parte del esfuerzo en el proceso de análisis y diseño de mejores prácticas. El *refactoring* es un proceso vital en cualquier proceso de desarrollo de calidad, pues este permite que se cometan errores que afecten la mantenibilidad durante etapas de desarrollo para corregirlos posteriormente, posibilitando así las nociones modernas de desarrollos y entregas rápidas, pero sin comprometer y a la vez incluso promoviendo la implementación de buenas prácticas, el constante mejoramiento y revisión de la calidad de software [1].

Para el mejoramiento del proceso de registro y documentación se llevan a cabo varias acciones enfocadas en la automatización de documentación a partir de procesos de control de versiones, de forma tal que la creación de documentación no incurriera en ser un proceso invasivo en el trabajo de desarrollo, pero aún así resultase en documentación clara y acertada

sobre el trabajo realizado. La primera parte de esta automatización incluye la implementación de *conventional commits*, lo cual es un estándar que se basa en agregar información extra a los comentarios realizados tras cada cambio de código, siendo dicha información definida por una simple palabra en un formato específico, lo cual permite a grandes rasgos reconocer si un cambio realizado implica la solución de *bugs* o la implementación de nuevas *features* [4]. Junto a esto, se integran herramientas extras para validar la estricta utilización de este estándar, garantizando así que todo cambio de código realizado cumpla con tener una apropiada partícula mínima de documentación. Y para extender incluso más el valor que se obtiene de esta pequeña implementación, se automatiza una tarea de creación de registros de versión, la cual consiste de generar una porción de documento llamado *changelog* el cual describe todos los cambios que se realizaron para cada versión, conteniendo la descripción de cada cambio y finalmente, adjuntando enlaces de redirección a todo tipo de espacio relevante para la revisión del cambió, como lo pueden ser las historias de usuario, los *pull request* o incluso al registro de individual de cada cambio.

Como último paso de las mejoras y el más grande en términos de trabajo manual está la migración de tecnología, la cual demostró ser el paso más complicado pues suponía el reto de conectar una librería de desarrollo *FrontEnd* moderno llamada *React*, con una librería *core* de la empresa la cual es imprescindible para la interacción y comunicación del negocio, pero qué, sin duda, ha sido el principal causante de los problemas de mantenibilidad debido a ser poco práctica en su utilización y representar múltiples retos al implementar nuevas funcionalidades o modificar las existentes. Así bien, la migración de tecnología apunta a reducir el impacto que tiene dicha librería sobre la complejidad de los procesos de desarrollo, logrando esto a través de la implementación de múltiples patrones de diseño que limiten la incumbencia de la librería *core* en el funcionamiento de la nueva tecnología. Este trabajo en grandes rasgos representa la creación de interfaces que encapsulan, extienden e incluso blindan contra errores, las funcionalidades ya existentes de la librería *core* y presentan puntos de acceso más amigables y fáciles de utilizar para evitar la interacción directa con dicha librería, permitiendo de esta forma que el uso de una tecnología moderna no se vea limitada por ninguna característica ajena a sí misma. Dicha implementación comprende la utilización de un extenso número de patrones, pero entre los que más resaltan se encuentran el patrón *Factory*, el cual define un objeto que se encarga de la creación de entidades de un tipo, para este caso, la inicialización de la librería *core*, de tal forma

que el desarrollador no interactúa directamente con la librería, sino con el componente *Factory*, y este último se encarga de proveer información sobre estado, resultado y manejar cualquier error que se pueda presentar durante el proceso de inicialización. Por otro lado se menciona a los patrones más importantes de este trabajo: *Facade* y *Adapter*, conceptos los cuales hacen posible la idea de "encapsular" las funcionalidades de la librería *core*, para que la integración de la nueva tecnología no deba incurrir en constantemente crear piezas de código que permitan la compatibilidad, pues dicha tarea es resuelta en la implementación de estos patrones al crear interfaces que permiten que ambos elementos del aplicativo "hablen el mismo idioma". Como último patrón a destacar, vale la pena mencionar el patrón de comportamiento *Observer*, el cual consiste en asignar a un objeto la responsabilidad de escuchar y reportar los cambios que suceden en un lugar específico de la aplicación. De esta forma, otros componentes pueden ser reportados del cambio sin tener que estar conectados al cambio mismo, sino al objeto "*Observador"*, reduciendo de esta forma la cantidad de acoplamiento presente en la aplicación para obtener un buen nivel de reactividad en los componentes de esta [5].

IV. METODOLOGÍA

Puesto que el proyecto poseía ciertas características particulares en el contexto del desarrollo del mismo, se tuvieron que considerar algunos aspectos que influyeron en la elección de la metodología de trabajo. En primera instancia se encuentra la densidad de trabajo que poseía la empresa, lo cual representó una limitación en la asignación de recursos humanos para el proyecto, así como también la limitación del alcance, los cual causó que el proyecto se viera reducido a una prueba de concepto para tomar decisiones a partir de esta. Por ende, el inicio del proyecto se basó en la metodología de Spike, la cual consiste en realizar una tarea de la cual no se tenga suficiente conocimiento y/o certeza de cómo llevar a cabo o de los escenarios a considerar. Y tras finalizar esta, se debe hacer una descripción detallada del proceso y los resultados, resaltando los casos particulares que pueden surgir y describiendo cómo se debe llevar a cabo la tarea para que funcione en distintos escenarios [6]. En el presente proyecto el proceso consistía en tomar la aplicación existente y hacer un recorrido por el ciclo de ejecución normal de la aplicación. Durante este recorrido se hacía registro de los bloques o archivos de código que se consideraba que no cumplían con los principales patrones de diseño y/o estándares de código limpio. Una vez identificadas dichas secciones, se debía definir de qué manera se iban a modificar o cómo se podrían recrear durante el desarrollo de la nueva aplicación. Finalmente, dependiendo del nivel de complejidad del código evaluado, se debía considerar si el desarrollo de la prueba de concepto era viable para demostrar cómo dicha sección lograba ser mejorada en términos de mantenibilidad al implementar los apropiados patrones de diseño e integrar dicha funcionalidad usando la nueva tecnología. Para el caso, este último proceso consistió en tomar la sección de código elegida, inicializar un proyecto con *React* e integrar la librería *core*. Con esto, demostrar que en primer lugar, la librería inicializa correctamente y posteriormente, validar que utilizarla se hace más sencillo en términos prácticos gracias un entorno de código más ordenado e interfaces más fáciles de implementar.

Tras haber presentado la prueba de concepto se evidencia la viabilidad del proyecto y la potencial oportunidad de que el producto final representa no solo una mejora en el aplicativo objetivo inicial sino también una herramienta bastante útil para la creación de proyectos futuros o la refactorización completa de otros proyectos existentes que presenten el mismo problema de mantenibilidad. Por lo que junto a la aprobación y formalización del proyecto se tomó una

metodología más adecuada para el estilo de trabajo que se manejaría: *Rapid Application Development* o RAD es una metodología que encaja perfectamente con el contexto de desarrollo de este proyecto, pues consiste en un ciclo constante de definición de requisitos, prototipado, integración, retroalimentación inmediata y revisión del alcance [7]. Siendo este ciclo beneficioso para el proyecto pues pese a que este presenta un alcance claro con respecto a los objetivos base, es un proyecto altamente extensible el cual puede tener su alcance modificado en pos de contemplar mejoras o implementación de elementos que faciliten el uso de este a futuro. Por otro lado, la retroalimentación es una parte esencial de esta metodología, donde el cliente hace comentarios sobre la funcionalidad desarrollada y suele solicitar modificaciones en caso de requerirse. Para el caso, puesto que el proyecto es de índole interno, el cliente que evalúa la efectividad de la tarea desarrollada ha sido un papel tomador por el resto de personas involucradas en el proyecto, proveyendo así retroalimentación a partir de su experiencia trabajando en el aplicativo y opinando si el desarrollo representa una mejora en los trabajos a futuro o si se requiere hacer algún cambio para contemplar otros escenarios en los cuales se han presentado problemas en el pasado. La última ventaja obtenida a partir de este ciclo de trabajo es la posibilidad de introducir nuevos requisitos en cualquier momento del proyecto, puesto que estos nunca buscarán afectar los requisitos ya completados, sino que solo extienden la implementación de mejoras hacia otras funcionalidades, permitiendo así que el desarrollo de la nueva aplicación signifique una mejora en todas las direcciones, sin tener que sufrir de modificaciones de las mejoras ya implementadas. En última instancia, vale la pena hacer la anotación de que pese a que RAD se considera una metodología que puede promover la aparición de mala calidad de código, lo cual iría en contra del propósito del proyecto, hay que tener en cuenta que el producto de este proyecto no es el aplicativo en sí, sino las mejores que se le hacen a la calidad de código del aplicativo, por lo que este aspecto negativo de la metodología se puede considerar erradicado al ser un punto de vital importancia que se evalúa durante todo el desarrollo del proyecto. Agregando también que la causa de dicha pérdida en la calidad de código es causada por la limitación de tiempo que puede llegar a presentarse en las fases de desarrollo, limitación la cual es casi inexistente para el proyecto en su totalidad debido a ser este de índole interno.

Para hacer seguimiento administrativo del proyecto se utilizó la herramienta de gestión de proyectos JIRA, la cual brinda la posibilidad de crear tareas denominadas *tickets*, asignarlas a un

responsable, definir una descripción para la tarea, un tiempo estimado de desarrollo, definir diferentes estados para la tarea, realizar comentarios y mantener un registro temporal de las modificaciones de esta. Dicha herramienta se utiliza al inicio del proyecto para crear un *ticket* por cada una de las funcionalidades principales de la aplicación existente a las que se espera dar tratamiento para mejorarlas en la nueva aplicación. El proceso de uso de JIRA es el siguiente: al crear un *ticket* se debe definir la tarea a la cual corresponde y describir el resultado que se espera obtener de la resolución de esta. Una vez creado el *ticket* se asigna al responsable y este cambia el estado del *ticket* a "En progreso" al empezar a trabajar en ella. Una vez la tarea es completada, se cambia su estado a "Revisión" y se asigna a otra persona para que evalúe la calidad del trabajo realizado. Si se considera que este cumple con los estándares esperados, se pasa el *ticket* a estado de "Aprobado" y se solicita la integración del trabajo al proyecto de código general. Caso contrario, se comentan las correcciones necesarias para aprobar la tarea y se devuelve el *ticket* al desarrollador para aplicar los cambios y luego realizar todo el proceso de nuevo. Tras finalizar todo el trabajo, se pasa el *ticket* a estado de "Cerrado" y la tarea se considera completada [8].

Respecto a las tecnologías que valen la pena mencionar por su implicación en el desarrollo de este proyecto se encuentran JavaScript como lenguaje de programación utilizado en toda la aplicación. Se implementa la librería *React* como la mencionada "tecnología moderna" para el desarrollo de aplicaciones FrontEnd, siendo esta una librería que permite el desarrollo de comportamientos reactivos, lo cual se refiere a la constante actualización de componentes con base en eventos resultantes de la modificación de variables en tiempo real. Por otro lado, es importante destacar la librería *core* de la empresa la cual se usa para manejar la lógica y datos de negocio que son creados y almacenados por el cliente, ofreciendo una interfaz para recuperar, modificar y crear información del producto enseñado en el aplicativo. Para el control de versiones del código se hace uso de la herramienta Git, la cual permite mantener registro de cada cambio que se realiza en el código y ofrecer múltiples acciones para modificar dichos cambios y analizar el histórico de estos. Dicha herramienta se usa en compañía de un repositorio virtual, por medio del cual se puede interactuar con otras personas para compartir y analizar la información creada por las múltiples iteraciones de registros de cambios de código, siendo dos los repositorios seleccionados para el trabajo, BitBucket y GitHub, uno para la publicación formal del trabajo y el otro para la publicación de las pruebas, respectivamente. Así mismo, se utilizan herramientas de integración y despliegue continuo en cada una, Circle CI y GitHub Actions para BitBucket y

GitHub respectivamente. Estas últimas son herramientas que permiten automatizar el proceso de despliegue de tal forma que no dependan de las acciones de un desarrollador, garantizando que tras cada integración de una pieza de trabajo, se pueda tener una versión desplegada en la web para hacer pruebas de esta online. Cabe recordar que como se mencionó anteriormente, se hace uso de una herramienta la cual permite generar documentación a partir del formato que se utiliza en la creación de comentarios en cada registro de versión, por lo que, como parte de la metodología es importante mencionar que el correcto nombramiento y formato de cada versión es importante para el desarrollo del proyecto.

En última instancia, el trabajo de desarrollo consistía de un proceso imperativo de análisis de la funcionalidad objetivo en el aplicativo existente y posterior integración en el nuevo aplicativo, siempre buscando asegurar la implementación de estándares de código limpio, tales como la implementación de patrones de diseño adecuados para el caso de uso y el uso de convenciones estandarizadas para nombrado de variables y funciones. Tras cada iteración de desarrollo se crea la versión del código y se publica en el repositorio virtual junto al respectivo comentario con el adecuado comentario y formato para su correcta documentación automática. Una vez la versión está en el repositorio, se realiza la correcta actualización en el *ticket* de JIRA y se asigna a la persona encargada de realizar la revisión del trabajo y proceder con el ciclo de vida esperado del *ticket*.

V. RESULTADOS

Tras la evaluación del estado del aplicativo se termina identificando que este se caracteriza por la ausencia total de patrones de diseño y la falta de una arquitectura concisa a lo largo de toda la aplicación, la cual pese a ser un software con estructura modular, realmente falla en la aplicación del concepto debido a incurrir en presentar una muy baja cohesión evidente en varios módulos, conteniendo múltiples funcionales para utilidad, interacción y comportamiento ajenos a los propio dentro de un solo componente. Se diagnostica que esta aglomeración de funcionalidades pobremente cohesivas derivó también en un alto acoplamiento, principalmente generado por la necesidad de acceder funciones ubicadas en diferentes módulos. Junto a esto, la falta de una interfaz que permita conectar los componentes entre sí sin requerir una comunicación directa, termina demostrando demasiadas relaciones de dependencia entre los componentes del aplicativo.

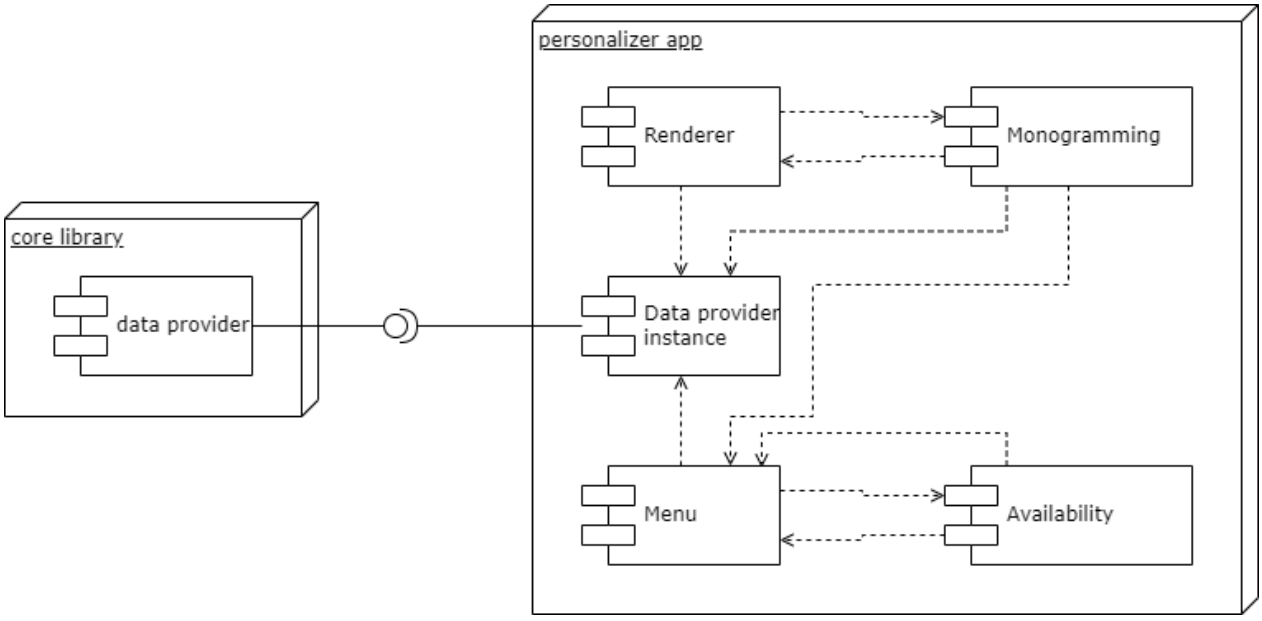

Fig. 1. Modelo de componentes del módulo de Proveedor de Datos.

Se eligió un conjunto de los módulos más esenciales y robustos de la aplicación para ser los principales a los cuales realizarles el tratamiento de rediseño e integración en la nueva tecnología: primero se encuentra el módulo del proveedor de datos, el cual se encarga en primer instancia de recibir parámetros (id de cliente, id de producto, localización, lenguaje, moneda) y utilizar estos para establecer la conexión al servidor de la empresa con el propósito de traer únicamente los datos relevantes para la sesión, y posteriormente proveer el canal de comunicación con un gestor de estado. Este módulo se diagnosticó como el más problemático, puesto que su responsabilidad no estaba limitaba a solo inicializar la comunicación, traer los datos y proveer la interfaz del gestor de estado como se esperaría, sino que se convirtió en el punto central de datos para que cualquier otro componentes se conecte a él en busca de obtener datos del producto y acceder a la interfaz. Por otro lado, durante la inicialización del módulo anterior se interrumpía su ejecución para hacer llamado a otro módulo que se encarga de traer información de disponibilidad de las opciones del producto. Este módulo fue el segundo seleccionado para tratar, pues en primera instancia y como se acaba de mencionar, presentaba un inconveniente de acoplamiento al llamarse justo durante la ejecución de otro módulo el cual a primera vista no debería tener relación directa con este. Además, la estructura de este módulo se ha presentado como poco flexible a modificaciones, puesto que en el pasado ha implicado grandes cantidades de trabajo para realizar pequeñas implementaciones que extiendan la funcionalidad del mismo. El tercer módulo a trabajar fue el de monogramas, este provee la lógica necesaria para renderizar un elemento de customización llamado “monograma”. La razón por la que se debe realizar tratamiento sobre el módulo es su gran densidad y poca cohesión, ya que este contiene piezas de lógica que comprenden funciones fuera de su responsabilidad como funciones para validar y modificar porciones de texto, con las cuales incurre en procesar también información proveída por otros módulos en pos de completar solo la tarea de renderizado. Como último módulo propuesto a trabajar se encuentra el generador de recetas, encargado de crear un resumen de información de la personalización que luego se debe enviar al entorno de *e-commerce* del cliente para proceder con la solicitud de la orden. El problema identificado en este módulo es muy similar al anterior, pues termina realizando demasiadas más tareas de las que le corresponde para poder completar su tarea principal. El generador de recetas accede a otros módulos para traer la información necesaria, ejecuta validaciones sobre esta información e incluso modifica la configuración actual del producto en algunos casos, para finalmente crear la receta y una vez

más, ir fuera de sus responsabilidades al realizar el envío de la receta al cliente, siendo todas estas funciones que no deberían realizarse dentro de este mismo módulo.

Una vez seleccionados los módulos e identificados sus problemas principales, se diseñó un plan para atender la necesidad de cada uno de ellos. El módulo de proveedor de datos requirió la solución más extensa y compleja de todas, la cual consistió en primero implementar el patrón *Factory* para la inicialización del objeto de comunicación y la entrega de los datos y de la interfaz de comunicación al gestor. De esta manera se logró la descentralización del módulo proveedor de datos, repartiendo las tareas entre otro par de módulos, uno encargado de proveer data y el otro encargado de ofrecer acceso al gestor de datos externo. Para este segundo módulo se realizó un trabajo extra, puesto que la interfaz del gestor de datos presenta su propia estructura y métodos, se decidió implementar un patrón *Facade* para que dichos métodos tuviesen un mejor acoplamiento y fuesen consistentes con los estándares de la aplicación nueva. Esta fue la porción de trabajo más densa del proceso, pues supuso el reto de encapsular todas y cada una de las funciones existentes en la interfaz del gestor de datos y en algunos casos, incluso extender las funcionalidades de las mismas para que se acomodaran mejor a las necesidades actuales del contexto de desarrollo. El último paso que se dió para mejorar el diseño de este módulo fue crear un gestionador de datos local, al cual pueda acceder fácilmente cualquier otro componente en el aplicativo y por medio del patrón *State* (el cuál es facilitado por la nueva tecnología utilizada, React) se asegura que dichos componentes siempre se actualicen cuando haya un cambio de estado en los datos proveídos por el gestionador. El módulo de disponibilidad fue modificado para implementar un patrón *Adapter*, lo cual abrió las posibilidades a que el módulo mismo no deba ser modificado si la información de disponibilidad debe ser solicitada a otro servidor, sino que solo se debe crear otro componente que solicite la información de forma distinta y conectar este componente en vez del anterior. De esta manera, el adaptador se encarga de procesar la información recibida y de enviarla en el formato requerido de vuelta al componente que procesa la información a las necesidades del aplicativo. Como se infiere de lo explicado anteriormente, las responsabilidades del módulo de disponibilidad terminaron repartidas entre tres componentes distintos, un componente “conectable y desconectable” que se encarga de proveer los datos de disponibilidad al traerlos desde algún servidor, un componente adaptador que solo formatea la información y un componente de procesamiento que se encarga de realizar todo el trabajo pesado con los datos antes de alimentar el aplicativo con ellos. El módulo de monograma no requirió la

implementación de un patrón de diseño en particular pero sí vió su estructura modificada de tal forma que la recuperación y manejo de datos no tenga que ser realizado por él mismo sino que le sean otorgados por un nuevo componente intermediario. Sin embargo, se termina implementando el patrón *Adapter*, no como una necesidad para la mejora de la calidad, sino como una mejora del aplicativo mismo en pos de que a futuro, en caso de que se requiera reemplazar el módulo de monograma, la transición sea fácil y directa. Finalmente, el módulo de creación de receta recibió un cambio en estructura para poder implementar el patrón *Mediator*. Primero se aisló la lógica del creador de receta para que su responsabilidad consistiera únicamente de recibir los datos necesarios previamente procesados y generar el objeto de receta para retornarlo. La lógica de solicitud y procesamiento de datos se movió a otro componente y la lógica para enviar los datos al cliente se movió a su propio componente también. Tras tener toda esta lógica aislada en sus propias secciones, se requería un componente central que haga de mediador, de tal manera que todos ellos no deben comunicarse entre sí sino solicitar la información y las acciones al componente central, el cual se encarga de reactivamente interactuar con el resto de componentes de ser necesario.

Módulo	**Proveedor de datos**	**Disponibilidad**	**Monograma**	**Receta**
Función	Inicializar la comunicación con el servidor para proveer los datos y una interfaz de interacción con un gestor de estado.	Solicitar datos de disponibilidad en inventario a un servidor externo y proveerlos al aplicativo.	Renderizar un diseño personalizado a partir de texto.	Generar un objeto de datos que resuma las selecciones de personalización realizadas por el usuario.
Problema	Alta centralización y dependencia en este módulo. Además de ser el punto de acceso para funcionalidades fuera de su responsabilidad.	Incorrecta integración del módulo y baja cohesión al cumplir con más responsabilidades de las apropiadas por encargarse también del procesamiento de datos.	Baja cohesión al incurrir en procesamiento y recuperación de datos necesarios para la renderización.	Alto acoplamiento a múltiples módulos y baja cohesión al encargarse del procesamiento de los datos necesarios e incluso incurrir en modificación de estados de la aplicación.
Tr	● Distribuir las responsabilidades	● Distribuir las responsabilidades	● Distribuir las responsabilidades	● Distribuir las responsabilidades

atamiento	del módulo entre múltiples componentes. ● Crear un componente para implementar el patrón *Factory* en la inicialización del proveedor de datos. ● Crear un componente para implementar el patrón *Facade* en la interacción con la interfaz. ● Crear un componente para implementar un gestor de estado local con el patrón *State* y reducir la dependencia en la interfaz del gestor de datos.	del módulo entre múltiples componentes. ● Crear un componente para implementar el patrón *Adapter* y minimizar el impacto de cambios en el solicitador de disponibilidad.	del módulo entre múltiples componentes. ● Crear un componente para implementar el patrón *Adapter* y reducir el impacto de tener que reemplazar el renderizador.	del módulo entre múltiples componentes. ● Crear un componente para implementar el patrón *Mediator* y aumentar el nivel de cohesión que tiene el módulo a la vez que se evita generar alto acoplamiento de los componentes creados para distribuir las tareas.

Fig. 2. Soluciones propuestas para los problemas de cada módulo.

Al finalizar el diseño de todo el trabajo principal del proyecto se inició con el desarrollo de este, para el cual se llevaron a cabo revisiones semanales sobre los avances y las integraciones que se realizaban durante dicha semana. Durante estos avances se propusieron diferentes extensiones a las funcionalidades del módulo de gestión de datos como foco principal. Dichas modificaciones nacían de la necesidad de facilitar procesos que usualmente se repetían bastante durante algunos procesos de desarrollo. Escenarios tales como siempre tener que formatear la misma porción de datos de alguna forma, mapear colecciones, filtrar con base en estados específicos, entre otros. Todas estas funciones se fueron implementando durante el proceso de desarrollo y se registraron como tareas de "*backlog*" para reducir el impacto que podían tener en el cumplimiento de las tareas principales. Tras finalizar el trabajo de migración del primer módulo: el proveedor de datos, la herramienta se notaba bastante robusta y muy adecuada para uso generalizado, por lo que la empresa decidió extender el alcance del proyecto. Teniendo en cuenta que todas las aplicaciones de la empresa hacen uso de la misma librería *core* que se ha mencionado múltiples veces, el nuevo aplicativo se presentaba hasta este punto como un

encapsulamiento completo de esta librería para ser integrada con *React*, lo que quiere decir que este acercamiento podría ser analogado por otros aplicativos que también requirieron la librería *core* y una migración a tecnologías modernas. Así bien, se toma la decisión de tomar el trabajo hasta este punto y guardarlo como una "plantilla" para la creación de nuevos aplicativos, de tal forma que todo el trabajo inicial de crear la estructura base de la aplicación ya se pudiera tener hecho. Este trabajo incluso evolucionó aún más hasta el punto en el que se creó un paquete ejecutable de *NPM* el cual al ser llamado crea la base de código con el gestor de datos y con todos los componentes básicos que se puedan requerir.

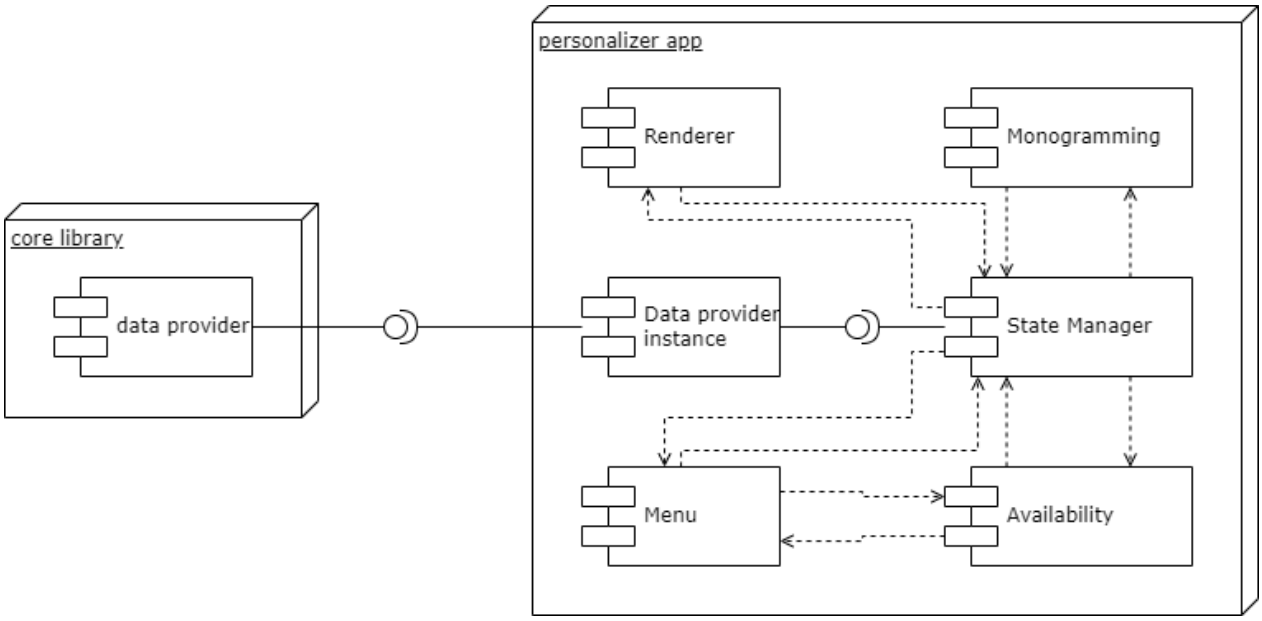

Fig. 3. Modelo de componentes de la implementación del patrón *State* en el módulo Proveedor de Datos.

VI. IMPLEMENTACIÓN

Analizando el primer paso del proyecto: la identificación de los problemas del aplicativo, el reto allí fue poder enfocar los esfuerzos en puntos esenciales de la aplicación de forma tal que se pudiera tener un plan concreto sobre los objetivos mínimos a cumplir para finalizar el proyecto académico incluso si el proyecto laboral continuará expandiéndose, lo cual fue el caso. La identificación de los módulos vitales de la aplicación fue bastante acertada pues al concluir el desarrollo del proyecto fueron relativamente pocos los elementos que no tuvieron cobertura de mejoramiento pero que aún así no afectaron el funcionamiento de este. Con esto se aseguró cumplir con el objetivo de diagnosticar los fallos principales del aplicativo, dando paso a la definición de buenas prácticas que se deberían considerar para el desarrollo de la nueva aplicación.

La definición de los patrones a aplicar y su respectiva implementación fue una fase que se extendió de forma paralela hasta y durante toda la fase de desarrollo, pues gracias al constante feedback que se tenía, la aplicación estuvo extendiendo sus implementaciones más y más con el propósito de proveer un mejor control de escenarios de modificación y de la misma forma, mejorar aún más la mantenibilidad que esta aplicación podría tener frente a grandes implementaciones o modificaciones masivas.

Durante el desarrollo no hubieron retos particulares y el proceso fue relativamente directo, sin embargo, es importante mencionar que el aplicativo adquirió niveles de complejidad más alto, pues al mejorar la mantenibilidad de un proyecto a través de cambiar su arquitectura, estandarizar la estructura e implementar múltiples patrones de diseño, se integra más código y más complejo del que se necesitaría para tener una funcionalidad que sea poco mantenible. Sin embargo, este inconveniente no es más que una preocupación verbal, pues la intención de agregar bloques de código más complejos no es convertir la aplicación en una más compleja de modificar, ya que dicho código se crea de tal manera que no deba ser modificado en caso de que se requieran nuevas implementaciones.

Como última parte del análisis es de vital importancia destacar el éxito de la implementación con respecto a su futura usabilidad en CreateMe, púes como se mencionó, al extrapolar esta mejora de un aplicativo ya existente, para que sea una herramienta de creación de aplicaciones base, dicha herramienta apunta a convertirse un estándar para el desarrollo de

proyectos dentro de la empresa, representando una reducción considerable en el tiempo de desarrollo requerido para llevar a cabo el desarrollo de la estructura inicial de cualquier aplicación, pues este proceso siempre suele ser un trabajo repetitivo y similar entre todos los proyectos de CreateMe. En términos de estimación, el desarrollo de la base de un aplicativo en la empresa corresponde a un valor de hasta 60 horas, que incrementa entre un 30% a un 50% al agregar tiempos administrativos, representando un costo total de hasta 90 horas de desarrollo en proyectos que suelen tomar entre 600 horas hasta 1200 horas, lo cual significa finalmente entre el 8% hasta el 16% de ganancias extras por proyecto al eliminar dichos tiempos de desarrollo, valor el cual solo seguirá incrementando en cuanto se pueda seguir implementando más funcionalidades básicas a la herramienta de creación de aplicativos.

VII. CONCLUSIONES

Realizar un Spike para adquirir el suficiente conocimiento sobre el resto del aplicativo fue la decisión más acertada que se pudo tomar, pues esto no significó solamente la aprobación del proyecto sino que sentó las bases teóricas para llevar a cabo la implementación de mejoras en cada uno de los módulos seleccionados para tratar, permitiendo reducir los tiempos de desarrollo que tuvo cada tarea definida y minimizando la aparición de inconvenientes debido a escenarios no considerados en un principio.

Los patrones definidos a ser implementados desde un inicio fueron bastante adecuados pues lograron cumplir con el objetivo de eliminar gran parte de la deuda técnica e incluso prevenir la aparición de deuda técnica futura, esto último siendo soportado también por las implementaciones extras que se realizaron con respecto a la documentación automática del proceso de desarrollo, dejando una trazabilidad clara del contenido cada uno de los registros.

El desarrollo de la aplicación no representó un gran reto técnico pues en el pasado se había intentado algo similar y ya se poseían ciertas nociones técnicas de qué prácticas de desarrollo evadir y qué caminos tomar con respecto a la implementación, sin embargo, debido al enfoque nuevo que se tenía con respecto a maximizar la mantenibilidad del código, el reto principal fue asegurar que las implementaciones de cada sección de código refinado tuvieran la mejor calidad posible y presentaran beneficios de mantenibilidad y extensibilidad.

Cambiar el propósito final del proyecto fue una excelente elección, pues convirtió un proyecto que apuntaba ser una mejora interna para un aplicativo existente, en una herramienta excelente que incluso muchos dentro de la empresa anhelaban tener desde hace tiempo, herramienta la cual no solo representa una mejora en el aplicativo original al que apuntaba el presente proyecto, sino también una potencial mejora para otros aplicativos similares y una prevención para la aparición de este tipo de aplicaciones a futuro.

Dicho todo esto, se asegura que los objetivos planteados para el proyecto se lograron de forma efectiva y sobresaliente, pues logra cumplir con todas las expectativas de este proyecto académico y también logra el objetivo de todo proyecto industrial interno: mejorar de forma significativa las capacidades de la empresa.

VIII. RECOMENDACIONES

Los rediseños y migraciones de cualquier tipo de aplicativo son un proceso extensamente costoso, tanto en tiempo como en recursos. Por ende, es altamente recomendable evitarlas a toda costa, lo cual se puede lograr al considerar estos procesos como un costo a largo plazo al momento de estimar el esfuerzo de desarrollo de cualquier proyecto. Ocasionalmente se tiende a recortar tiempos de desarrollo durante una estimación, bien sea con el propósito de entregar un producto más rápido o simplemente proveer una oferta más atractiva para un cliente. Sin embargo, considerar los costos de resolución de deudas técnicas en un proyecto que aspire tener cierta edad es una práctica favorable para la empresa, puesto que de tener que aplicarse este proceso, dichos costos deben ser asumidos por la empresa proveedora y no por el cliente, finalmente, convirtiendo el recorte de dos meses de desarrollo en una remodelación total de hasta medio año.

Cuando se habla de la implementación de mejoras de diseño en una aplicación de software, se habla de un proceso que no puede ser medible de forma cuantitativa, sino puramente cualitativa y únicamente por medio de la experiencia de un desarrollador. Pues este tipo de procesos en realidad no influyen en ninguna métrica de valorización de las aplicaciones, significando nada más que una noción que va más allá del desarrollo mismo y que pese a su aplicación reside en el código, su efecto está en el negocio. Así bien, se invita al lector a que evalúe las posibilidades de diseñar e implementar métricas razonables para evaluar la calidad del diseño de una aplicación, permitiendo así que un desarrollador con cualquier nivel de experticia pueda realizar una evaluación razonable del diseño de su aplicación.

REFERENCIAS

[1] M. Fowler, “Refactoring: Improving the Design of Existing Code”. Addison-Wesley, 1999. [En línea]

[2] R. Martin, J. Coplien, “Clean code: a handbook of agile software craftsmanship”. Prentice Hall, 2009. [En línea]

[3] A. Hunt, D. Thomas, “The Pragmatic programmer : from journeyman to master”. Addison-Wesley, 2000. [En línea]

[4] ConventionalCommits ‘Conventional Commits’’. ConventionalCommits Web Docs, Accedido:20, Sept. 2024. [En línea]. Disponible en: https://www.conventionalcommits.org/en/v1.0.0/

[5] Gamma, E., et al, “Design Patterns: Elements of Reusable Object-Oriented Software”. Addison-Wesley Professional, 1994. [En línea]

[6] H. Al Hashimi and A. Gravell, "Spikes in Agile Software Development: An Empirical Study," 2020 *International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, USA, 2020, pp. 1715-1721. [En línea]

[7] P. Beynon-Davies, C. Carne, H. Mackay y D. Tudhope. “Rapid application development (RAD): an empirical review” in European Journal of Information Systems, vol. 8, no. 3, pp. 211-223, 1999. [En línea]

[8] A. Arnautović. "Managing project using JIRA software," in Serbian Journal of Engineering Management, vol. 7, pp. 40-46, 2022. [En línea]